NASA Procedural Requirements

COMPLIANCE IS MANDATORY

NPR 7150.2A
Effective Date:
November 19, 2009
Expiration Date:
November 19, 2014

Printable Format (PDF)

Request Notification of Change (NASA Only)

Subject: NASA Software Engineering Requirements

Responsible Office: Office of the Chief Engineer

| TOC | Preface | Chapter1 | Chapter2 | Chapter3 | Chapter4 | Chapter5 | Chapter6 | AppendixA | AppendixB | AppendixC | AppendixD | AppendixE | ALL |

Chapter 3: Software Engineering Life-Cycle Requirements

This NPR makes no recommendation for a specific software life-cycle model. Each has its strengths and weaknesses, and no one model is best for every situation. Whether using the spiral model, the iterative model, waterfall, or any other development life-cycle model, each has steps of requirements, design, implementation, testing, release to operations, maintenance, and retirement. Although this NPR does not impose a particular life-cycle model on each software project, this NPR does support a standard set of life-cycle phases. Use of the different phases of a life cycle allows the various products of a project to be gradually developed and matured from initial concepts through the fielding of the product and to its final retirement. Without recommending a life cycle, the requirements for each of these steps are provided below.

3.1 Software Requirements

The requirements phase is one of the most important phases of software engineering. Studies show that the top problems in the software industry are due to poor requirements elicitation, inadequate requirements specification, and inadequate management of changes to requirements. Requirements provide the foundation for the entire life cycle as well as for the software product. Requirements also provide a basis for planning and estimating. Requirements are based on customer, user, and other stakeholder needs and design and development constraints. The development of requirements includes elicitation, analysis, documentation, verification, and validation. Ongoing customer validation of the requirements to ensure the end products meet the customer needs is an important part of the life-cycle process. This can be accomplished

via rapid prototyping and customer-involved reviews of iterative and final software requirements.

3.1.1 Requirements Development.

3.1.1.1 The project shall document the software requirements. [SWE-049]

Note: The requirements for the content of a Software Requirement Specification and a Data Dictionary document are defined in Chapter 5. The requirements definition activity also includes documenting key decisions, developing requirement rationales, and defining assumptions. The requirement development activities can use lessons learned in performing the logical decomposition process activities. The requirements definition activity provides an understanding of the derived technical requirements baseline, a logical decomposition model, traceability to technical requirements, and an understanding of the stakeholder's expectations.

3.1.1.2 The project shall identify, develop, document, approve, and maintain software requirements based on analysis of customer and other stakeholder requirements and the operational concepts. [SWE-050]

3.1.1.3 The project shall perform software requirements analysis based on flowed-down and derived requirements from the top-level systems engineering requirements and the hardware specifications and design. [SWE-051]

Note: The software requirements analysis determines the requirement's safety criticality, correctness, consistency, clarity, completeness, traceability, feasibility, verifiability, and maintainability. The software requirements analysis activities include the allocation of functional, non-functional, and performance requirements to functions and subfunctions.

3.1.1.4 The project shall perform, document, and maintain bidirectional traceability between the software requirement and the higher-level requirement. [SWE-052]

3.1.2 Requirements Management.

3.1.2.1 The project shall collect and manage changes to the software requirements. [SWE-053]

Note: The project analyzes and documents changes to requirements for cost, technical, and schedule impacts.

3.1.2.2 The project shall identify, initiate corrective actions, and track until closure inconsistencies among requirements, project plans, and software products. [SWE-054]

3.1.2.3 The project shall perform requirements validation to ensure that the software will perform as intended in the customer environment. [SWE-055]

Note: Requirements validation includes confirmation that the requirements meet the needs and expectations of the customer. Requirement validation is confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled.

3.2 Software Design

Software design is the process of defining the software architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements. The software architecture is the fundamental organization of a system embodied in its

components, their relationships to each other and to the environment, and the principles guiding its design and evolution. The software architectural design is concerned with creating a strong overall structure for software entities that fulfill allocated system and software-level requirements. Typical views captured in an architectural design include the decomposition of the software subsystem into design entities, computer software configuration items (CSCI), definitions of external and internal interfaces, dependency relationships among entities and system resources, and finite state machines. Detailed design further refines the design into lower-level entities that permit the implementation by coding in a programming language. Typical attributes that are documented for lower-level entities include: identifier, type, purpose, function, constraints, subordinates, dependencies, interface, resources, processing, and data. Rigorous specification languages, graphical representations, and related tools have been developed to support the evaluation of critical properties at the design level. Projects are encouraged to take advantage of these improved design techniques to prevent and eliminate errors as early in the life cycle as possible.

3.2.1 The project shall document and maintain the software design. [SWE-056]

Note: The requirement for the content of a Software Design Description document and an interface design description document are defined in Chapter 5 as applicable by software classification.

3.2.2 The project shall transform the allocated and derived requirements into a documented software architectural design. [SWE-057]

Note: The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the properties of those components, and the relationships between them. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects.

3.2.3 The project shall develop, record, and maintain a detailed design based on the software architectural design that describes the lower-level units so that they can be coded, compiled, and tested. [SWE-058]

3.2.4 The project shall perform and maintain bidirectional traceability between the software requirements and the software design. [SWE-059]

3.3 Software Implementation

Software implementation consists of implementing the requirements and design into code, data, and documentation. Software implementation also consists of following coding methods and standards. Unit testing is also usually a part of software implementation (unit testing can also be conducted during the testing phase).

3.3.1 The project shall implement the software design into software code. [SWE-060]

3.3.2 The project shall ensure that software coding methods, standards, and/or criteria are adhered to and verified. [SWE-061]

3.3.3 The project shall ensure that results from static analysis tool(s) are used in verifying and validating software code. [SWE-135]

Note: Modern static code analysis tools can identify a variety of issues and problems, including but not limited to dead code, non-compliances with coding standards, security vulnerabilities, race conditions, memory leaks, and redundant code. Typically, static analysis tools are used to help verify adherence with coding methods, standards, and/or criteria. While false positives are an acknowledged shortcoming of static analysis tools, users can calibrate, tune, and filter results to make effective use of these tools. Software peer reviews/inspections of code items can include reviewing the results from static code analysis tools.

Note: Static analysis tools may not be readily available for some platforms or computing languages. If static analysis tools are determined to not be available, the project can document the alternate manual methods and procedures to be used to verify and validate the software code. These manual methods and procedures will be addressed or referenced in the project's compliance matrix against this requirement.

3.3.4 The project shall ensure that the software code is unit tested per the plans for software testing. [SWE-062]

3.3.5 The project shall provide a Software Version Description document for each software release. [SWE-063]

Note: The requirement for the content of a Software Version Description document is defined in Chapter 5.

3.3.6 The project shall provide and maintain bidirectional traceability from software design to the software code. [SWE-064]

3.3.7 The project shall validate and accredit software tool(s) required to develop or maintain software. [SWE-136]

Note: Projects need to determine and define acceptance processes for software tool(s), used to develop or maintain Class A, B, C, or safety-critical software. Examples of software tools include but are not limited to compilers, code-coverage tools, development environments, build tools, user interface tools, debuggers, and code generation tools.

3.4 Software Testing

The purpose of testing is to verify the software functionality and remove defects. Testing verifies the code against the requirements and the design to ensure that the requirements are implemented. Testing also identifies problems and defects that are corrected and tracked to closure before product delivery. Testing also validates that the software operates appropriately in the intended environment.

3.4.1 The project shall establish and maintain: [SWE-065]

a. Software Test Plan(s).

b. Software Test Procedure(s).

c. Software Test Report(s).

Note: The requirements for the content of a Software Test Plan, Software Test Procedure, and Software Test Report are defined in Chapter 5.

3.4.2 The project shall perform software testing as defined in the Software Test Plan.

[SWE-066]

Note: A best practice for Class A, B, and C software projects is to have formal software testing conducted, witnessed, and approved by an independent organization outside of the development team. Testing could include software integration testing, systems integration testing, validation testing, end-to-end testing, acceptance testing, white and black box testing, decision and path analysis, statistical testing, stress testing, performance testing, regression testing, qualification testing, simulation, and others. Use of automated software testing tools are also to be considered in software testing. Test breadth and accuracy can be increased through the use of test personnel independent of the software design and implementation teams, software peer reviews/inspections of Software Test Procedures and Software Test Results, and employing impartial test witnesses.

3.4.3 The project shall ensure that the implementation of each software requirement is verified to the requirement. [SWE-067]

3.4.4 The project shall evaluate test results and document the evaluation. [SWE-068]

3.4.5 The project shall document defects identified during testing and track to closure. [SWE-069]

3.4.6 The project shall verify, validate, and accredit software models, simulations, and analysis tools required to perform qualification of flight software or flight equipment. [SWE-070]

Note: Center processes address issues such as numerical accuracy, uncertainty analysis, and sensitivity analysis, as well as verification and validation for software implementations of models and simulations. Information regarding specific verification and validation techniques and the analysis of models and simulations can be found in the NASA standard NASA-STD-7009.

3.4.7 The project shall update Software Test Plan(s) and Software Test Procedure(s) to be consistent with software requirements. [SWE-071]

3.4.8 The project shall provide and maintain bidirectional traceability from the Software Test Procedures to the software requirements. [SWE-072]

3.4.9 The project shall ensure that the software system is validated on the targeted platform or high-fidelity simulation. [SWE-073]

Note: Typically, a high-fidelity simulation has the exact processor, processor performance, timing, memory size, and interfaces as the flight unit.

3.5 Software Operations, Maintenance, and Retirement

Planning for operations, maintenance, and retirement must be considered throughout the software life cycle. Operational concepts and scenarios are derived from customer requirements and validated in the operational or simulated environment. Software maintenance activities sustain the software product after the product is delivered to the customer until retirement.

3.5.1 The project shall document the software maintenance plans in a Software Maintenance Plan document. [SWE-074]

Note: The requirement for the content of a Software Maintenance Plan is defined in Chapter 5.

3.5.2 The project shall plan software operations, maintenance, and retirement activities. [SWE-075]

3.5.3 The project shall implement software operations, maintenance, and retirement activities as defined in the respective plans. [SWE-076]

3.5.4 The project shall complete and deliver the software product to the customer with appropriate documentation to support the operations and maintenance phase of the software's life cycle. [SWE-077]

Note: Delivery includes, as applicable, Software User's Manual (as defined in Chapter 5), source files, executable software, procedures for creating executable software, procedures for modifying the software, and a Software Version Description. Open source software licenses are reviewed by the Center's Chief of Patent/Intellectual Property Counsel before being accepted into software development projects. Other documentation considered for delivery includes:

a. Summary and status of all accepted Change Requests to the baselined Software Requirements Specifications.

b. Summary and status of all major software capability changes since baselining of the Software Design Documents.

c. Summary and status of all major software tests (including development, verification, and performance testing).

d. Summary and status of all Problem Reports written against the software.

e. Summary and status of all software requirements deviations and waivers.

f. Summary and status of all software user notes.

g. Summary and status of all quality measures historically and for this software.

h. Definition of open work, if any.

i. Software configuration records defining the verified and validated software, including requirements verification data (e.g., requirements verification matrix).

j. Final version of the software documentation, including the final Software Version Description document(s).

k. Summary and status of any open software-related risks.

3.5.5 The project shall deliver to the customer the as-built documentation to support the operations and maintenance phase of the software life cycle. [SWE-078]

| NODIS Library | Program Formulation(7000s) | Search |

DISTRIBUTION:
NODIS